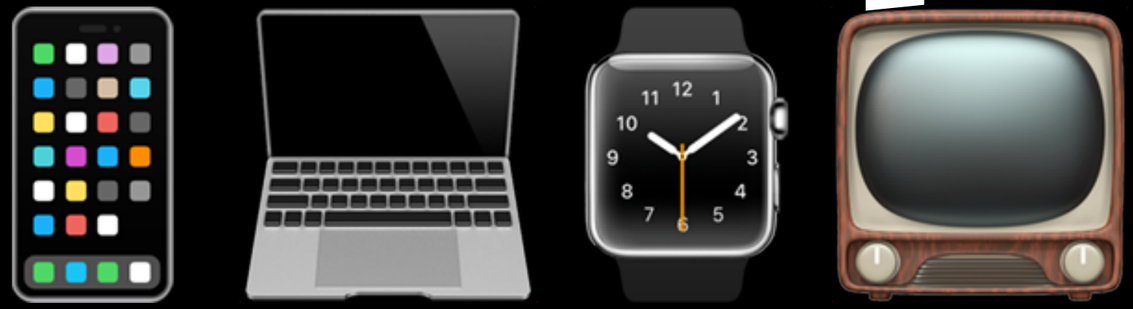
Making your own tool using SwiftSyntax


Hi, I'm Yusuke
@kitasuke

App development



Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Source code

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Tools 

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Tools help make programming easier

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

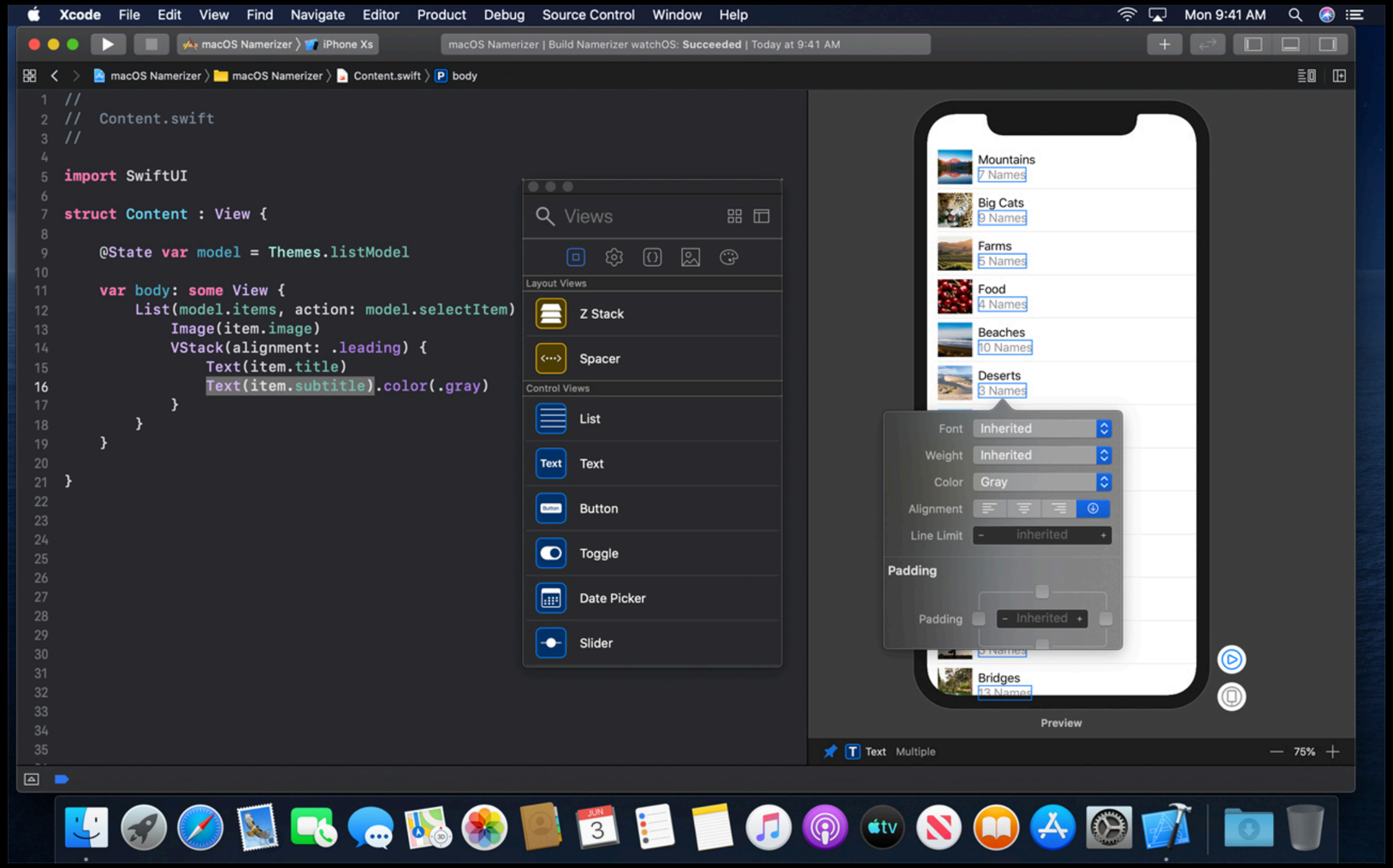
Tools using SwiftSyntax

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SwiftUI Previews

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SwiftUI Previews



Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

swift-format

SE-0250: Swift Code Style Guidelines and Formatter

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

swift-format

✅

```
let number: Int = 5
```

⛔

```
let number:Int = 5
let number : Int = 5
```

SwiftConst

Find in repeated strings that could be replaced by a constant

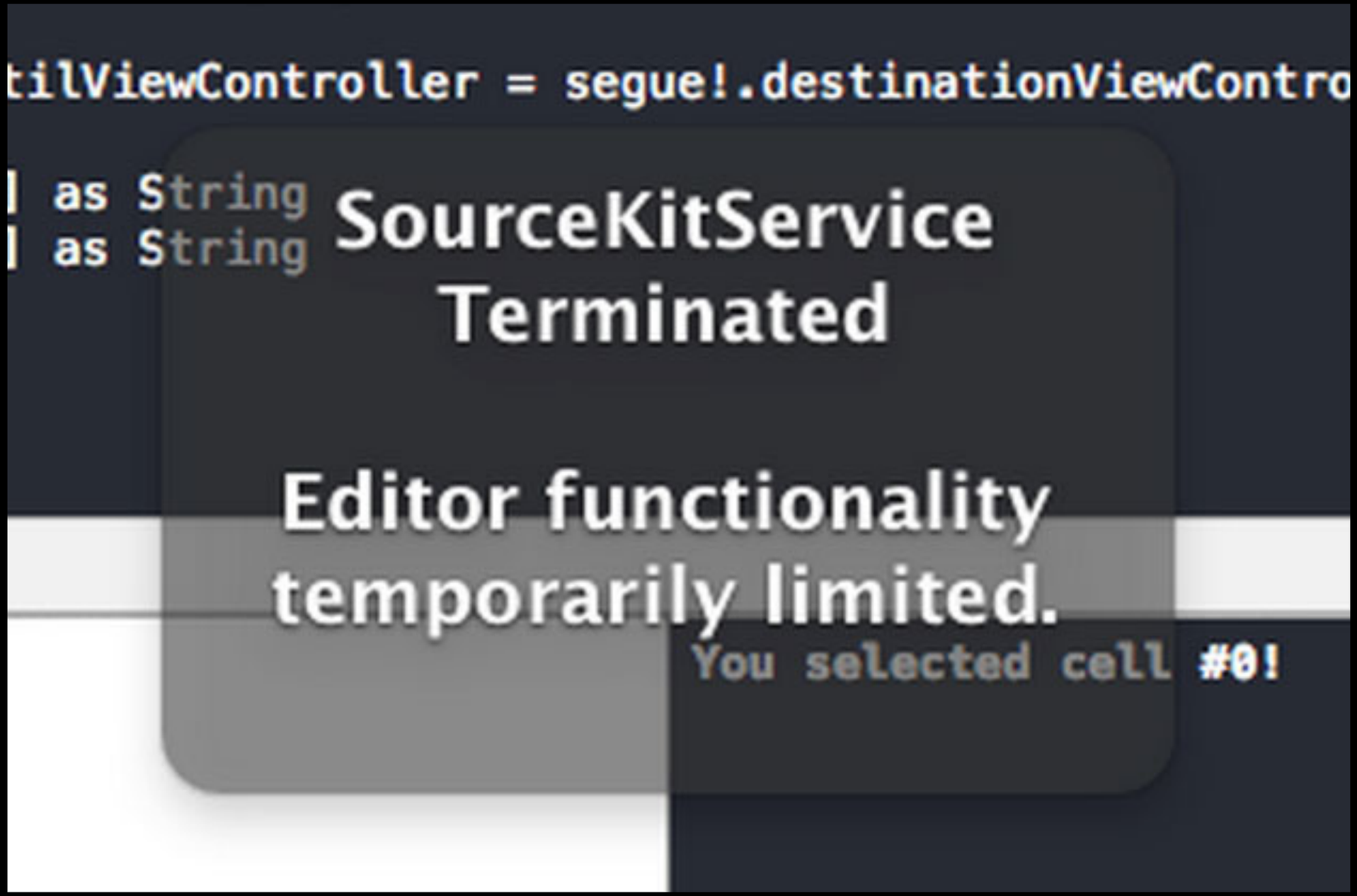
Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SwiftConst

```
$ swiftconst run
other occurrence(s) of "error" found in: main.swift:7:11
other occurrence(s) of "help" found in: main.swift:18:19
other occurrence(s) of "error" found in: main.swift:19:28
other occurrence(s) of "help" found in: main.swift:21:19
```

SwiftSyntax

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)


tilViewController = segue!.destinationViewContro
] as String
] as String
SourceKitService
Terminated
Editor functionality
temporarily limited.
You selected cell #0!

What's SwiftSyntax for?

Code modifier

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Code analyzer

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

What's SwiftSyntax?

SwiftSyntax is a set of Swift bindings for the libSyntax library. It allows for Swift tools to parse, inspect, generate, and transform Swift source code.

libSyntax

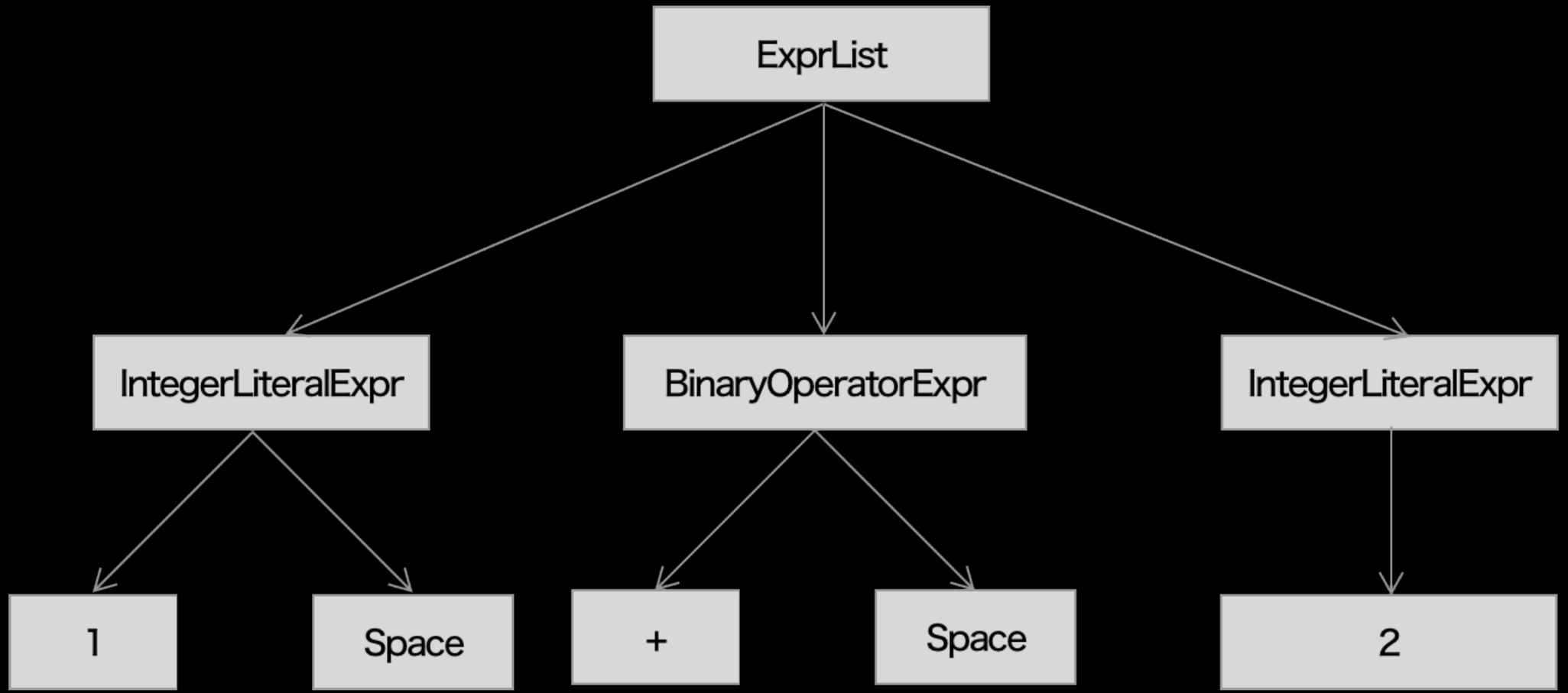
Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

libSyntax library aims to represent the syntax tree of the source file

Syntax Tree

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

1 + 2


ExprList
IntegerLiteralExpr
BinaryOperatorExpr
IntegerLiteralExpr
1
Space
+
Space
2

SwiftSyntax parses Swift source code to syntax tree

How to emit syntax tree

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

```
$ swift -frontend emit-syntax
```

Parse input file(s) and emit the Syntax tree(s) as JSON

$ swift -frontend emit-syntax

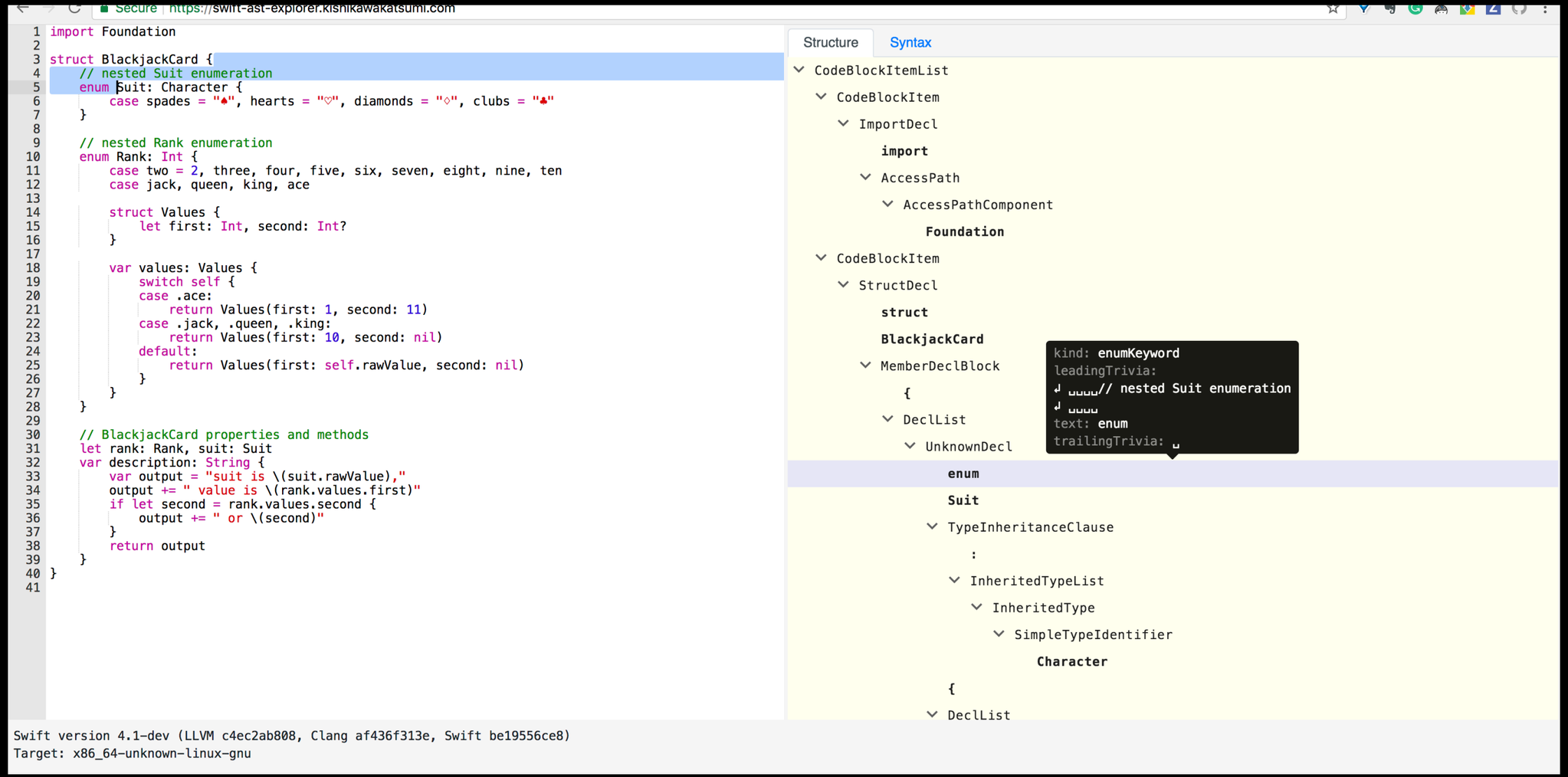
```
// 1 + 2
- SourceFile
    - CodeBlockItemList
        - CodeBlockItem
            - SequenceExpr
                - ExprList
                    - IntegerLiteralExpr
                        - integer_literal: 1
                        - trailingTrivia: Space
                    - BinaryOperatorExpr
                        - oper_binary_spaced: +
                        - trailingTrivia: Space
                    - IntegerLiteralExpr
                        - integer_literal: 2
```

Swift AST Explorer

Swift AST to HTML conversion

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Swift AST Explorer



Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Recap

Recap

→ SwiftSyntax is a parsing tool

Recap

→ SwiftSyntax is a parsing tool

→ Emits syntax tree

Recap

→ SwiftSyntax is a parsing tool

→ Emits syntax tree

→ Allows to inspect and transform source code

How SwiftSyntax works?

Usage via SwiftPM

Package.swift

```
import PackageDescription

let package = Package(
  name: "MyTool",
  dependencies: [
    .package(url: "https://github.com/apple/swift-syntax.git",
      .exact("<#Specify Release tag#>")),
  ],
  targets: [
    .target(name: "MyTool", dependencies: ["SwiftSyntax"]),
  ]
)
```

Interfaces

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SyntaxParser

Parsing Swift source

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SyntaxParser

```
/// Parses the file `URL` into a full-fidelity Syntax tree.
///
/// - Parameters:
///   - url: The file URL to parse.
///   - diagnosticEngine: Optional diagnostic engine to where the parser will
///       emit diagnostics
/// - Returns: A top-level Syntax node representing the contents of the tree,
///            if the parse was successful.
/// - Throws: `ParserError`
public static func parse(_ url: URL,
    diagnosticEngine: DiagnosticEngine? = nil) throws -> SourceFileSyntax
```

SourceFileSyntax

Structure of source file

SourceFileSyntax

```
// 1 + 2 in sample.swift
- SourceFileSyntax
    - CodeBlockItemListSyntax
        - CodeBlockItemSyntax
            - SequenceExprSyntax
                - ExprListSyntax
                    - IntegerLiteralExprSyntax
                        - integerLiteral: 1
                        - trailingTrivia: Space
                    - BinaryOperatorExprSyntax
                        - spacedBinaryOperator: +
                        - trailingTrivia: Space
                    - IntegerLiteralExprSyntax
                        - integerLiteral: 2
```

Syntax

IntegerLiteralExprSyntax

```
public struct IntegerLiteralExprSyntax:
  ExprSyntax, _SyntaxBase, Hashable {
  enum Cursor: Int {
    case digits
  }

  public var digits: TokenSyntax {...}
  ...
}
```

TokenSyntax

```
/// A Syntax node representing a single token.
public struct TokenSyntax: _SyntaxBase, Hashable {
  /// The text of the token as written in the source code.
  public var text: String {
    return tokenKind.text
  }
  ...
}
```

Syntaxes

`IdentifierExprSyntax`, `ArrayExprSyntax`,

`DictionaryExprSyntax`, `IntegerLiteralExprSyntax`,

`BooleanLiteralExprSyntax`, `FunctionCallExprSyntax`,

`StringLiteralExprSyntax`, `ClassDeclSyntax`, `StructDeclSyntax`,

`ProtocolDeclSyntax`, `VariableDeclSyntax`, `EnumDeclSyntax`,

`SwitchStmtSyntax`, `IfStmtSyntax` ...

SyntaxVisitor

Visiting each syntax

SyntaxVisitor

```
/// Visiting `IntegerLiteralExprSyntax` specifically.
///   - Parameter node: the node we are visiting.
///   - Returns: how should we continue visiting.
mutating func visit(_ node: IntegerLiteralExprSyntax) -> SyntaxVisitorContinueKind {
  return .visitChildren
}

/// The function called after visiting `IntegerLiteralExprSyntax` and its descendents.
///   - node: the node we just finished visiting.
mutating func visitPost(_ node: IntegerLiteralExprSyntax) {}
```

SyntaxRewriter

Rewriting each syntax

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SyntaxRewriter

```
open func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {
  return visitChildren(node) as! ExprSyntax
}
```

SyntaxFactory

Make each syntax

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

SyntaxFactory

```
public static func makeIntegerLiteralExpr(digits: TokenSyntax)
  -> IntegerLiteralExprSyntax {
  let layout: [RawSyntax?] = [
    digits.raw,
  ]
  let raw = RawSyntax.createAndCalcLength(
    kind: .integerLiteralExpr,
    layout: layout,
    presence: .present
  )
  let data = SyntaxData.forRoot(raw)
  return IntegerLiteralExprSyntax(data)
}
```

Usecases

Odd number inspector 

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Odd number inspector

```
// Use even instead, not odd
view.frame.height = 43
```

OddNumberInspector

```
struct OddNumberInspector: SyntaxVisitor {
    mutating func visitPost(_ node: IntegerLiteralExprSyntax) {
        // do nothing if it's even
        guard let integer = Int(node.digits.text),
            integer % 2 == 1 else {
            return
        }
        print("Use even instead, not odd")
    }
}
```

OddNumberInspector

```
struct OddNumberInspector: SyntaxVisitor {
    mutating func visitPost(_ node: IntegerLiteralExprSyntax) {
        // do nothing if it's even
        guard let integer = Int(node.digits.text),
            integer % 2 == 1 else {
            return
        }
        print("Use even instead, not odd")
    }
}
```

OddNumberInspector

```
struct OddNumberInspector: SyntaxVisitor {
    mutating func visitPost(_ node: IntegerLiteralExprSyntax) {
        // do nothing if it's even
        guard let integer = Int(node.digits.text),
            integer % 2 == 1 else {
            return
        }
        print("Use even instead, not odd")
    }
}
```

main.swift

```
// parse Swift source to get SourceFileSyntax
let sourceFile = try SyntaxParser.parse(pathURL)

// walk SourceFileSyntax with visitor
var visitor = OddNumberInspector()
sourceFile.walk(&visitor)
```

main.swift

```
// parse Swift source to get SourceFileSyntax
let sourceFile = try SyntaxParser.parse(pathURL)

// walk SourceFileSyntax with visitor
var visitor = OddNumberInspector()
sourceFile.walk(&visitor)
```

Odd number inspector

```
// Use even instead, not odd
view.frame.height = 43
```

Ten multiplication formatter 💪

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Ten multiplication formatter

```
let i: Int = 1 // -> 10
```

TenMultiplicationFormatter

```
class TenMultiplicationFormatter: SyntaxRewriter {
    override func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {
        guard let integer = Int(node.digits.text) else {
            return node
        }

        let digits = SyntaxFactory.makeIntegerLiteral(
            String(integer * 10),
            leadingTrivia: node.leadingTrivia ?? .zero,
            trailingTrivia: node.trailingTrivia ?? .zero
        )
        return IntegerLiteralExprSyntax {
            $0.useDigits(digits)
        }
    }
}
```

TenMultiplicationFormatter

```
class TenMultiplicationFormatter: SyntaxRewriter {
    override func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {
        guard let integer = Int(node.digits.text) else {
            return node
        }

        let digits = SyntaxFactory.makeIntegerLiteral(
            String(integer * 10),
            leadingTrivia: node.leadingTrivia ?? .zero,
            trailingTrivia: node.trailingTrivia ?? .zero
        )
        return IntegerLiteralExprSyntax {
            $0.useDigits(digits)
        }
    }
}
```

TenMultiplicationFormatter

```
class TenMultiplicationFormatter: SyntaxRewriter {
    override func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {
        guard let integer = Int(node.digits.text) else {
            return node
        }

        let digits = SyntaxFactory.makeIntegerLiteral(
            String(integer * 10),
            leadingTrivia: node.leadingTrivia ?? .zero,
            trailingTrivia: node.trailingTrivia ?? .zero
        )
        return IntegerLiteralExprSyntax {
            $0.useDigits(digits)
        }
    }
}
```

TenMultiplicationFormatter

```
class TenMultiplicationFormatter: SyntaxRewriter {
    override func visit(_ node: IntegerLiteralExprSyntax) -> ExprSyntax {
        guard let integer = Int(node.digits.text) else {
            return node
        }

        let digits = SyntaxFactory.makeIntegerLiteral(
            String(integer * 10),
            leadingTrivia: node.leadingTrivia ?? .zero,
            trailingTrivia: node.trailingTrivia ?? .zero
        )
        return IntegerLiteralExprSyntax {
            $0.useDigits(digits)
        }
    }
}
```

main.swift

```
// parse Swift source to get SourceFileSyntax
let sourceFile = try SyntaxParser.parse(pathURL)

// visit SourceFileSyntax
var visitor = TenMultiplicationFormatter()
let modifiedSourceFile = TenMultiplicationFormatter().visit(sourceFile)
```

main.swift

```
// parse Swift source to get SourceFileSyntax
let sourceFile = try SyntaxParser.parse(pathURL)

// visit SourceFileSyntax
var visitor = TenMultiplicationFormatter()
let modifiedSourceFile = TenMultiplicationFormatter().visit(sourceFile)
```

Ten multiplication formatter

```
let i: Int = 10 // <- 1
```

Tools make your code much better 😎

Making your own tool using SwiftSyntax, Yusuke Kita (@kitasuke)

Tools sometimes are better corder than you 😆

Improvements

→ Speeding up SwiftSyntax by using the parser directly

→ Integrating libSyntax into the compiler pipeline

→ Declarative syntax creation using function builders

Summary

Summary

→ Take benefit of statically typed language

Summary

→ Take benefit of statically typed language

→ Useful tools make better apps

Summary

→ Take benefit of statically typed language

→ Useful tools make better apps

→ SwiftSyntax's improving significantly

References

→ https://github.com/apple/swift-syntax

→ https://www.slideshare.net/kitasuke/integrating-libsyntax-into-the-compiler-pipeline

Thank you!